chapitre 13 : calcul matriciel

D.M. pour le 11 Mai 2014

Mode d'emploi : il s'agit d'un cours-D.M. Au fil du cours, vous allez trouver des questions qui pour la plupart demandent d'écrire des fonctions python mettant en oeuvre les algorithmes présentés. Merci de rédiger ce DM intégralement sur un fichier py (pour les questions complexité, taper le texte simplement). Merci de séparer les fonctions dans des cellules, avec le numéro de la question.

1 Le codage des matrices : PYTHON pur vs NUMPY

1.1 En python pur : on code une matrice par une liste de listes

Par exemple `A=[[1,2],[3,4]]` représentera pour nous la matrice $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$.

Les entrées de `A` sont obtenues via `A[i][j]`.

Noter que l'affichage d'une matrice n'est pas bien joli, mais qu'il est facile de programmer une petite fonction qui fasse un affichage joli.

Attention : En python, les indices commencent à 0, donc pour les matrices à `A[0][0]`.

1.2 Attention au préformatage pour les matrices en python pur

Une règle bien connue pour les listes :

On ne peut pas définir une liste ex-nihilo, en affectant des entrées, par exemple on ne peut pas définir la matrice du paragraphe précédent en posant `A[0][0]=1; A[0][1]=2;` etc...

On reverra pour les listes les différentes méthodes : `append`, concatenation `+` ou préformatage : pour une liste par exemple on peut créer une liste de zéros : `L=[0]*n` puis modifier les entrées, mais pour les listes de listes attention :

Danger

Un mauvais gag du préformage des listes doubles

On pourrait avoir envie de préformater une matrice 2×2 de zéros de la façon suivante :

```
>>> X=[[0,0]]*2
>>> X
[[0, 0], [0, 0]]
# jusqu'ici tout va bien.
>>> X[0][0]=1
>>> X
[[1, 0], [1, 0]]
```

On ne voulait modifier qu'une entrée et on a modifié deux ! D'où vient ce phénomène ?

Réponse : l'objet `[0,0]` qui est copié par l'opérateur `*2` est une liste. La copie pointe donc vers la même adresse mémoire (pointeur). Toute modification de la liste `X[0]` modifie aussi sa copie `X[1]`.

Comment contourner ce mauvais gag ?

a) D'abord en revoyant le chapitre sur les listes :
   • les listes ont beaucoup d'avantage. Pour une matrice fabriquée à partir de listes de listes, on peut modifier les entrées de la matrice, c'est bien, les fonctions peuvent transformer la matrice qu'elles ont en argument, c'est bien, mais il faut prendre garde aussi aux problèmes des éventuelles copies d'une matrice qui pointeraient vers la même case mémoire... une modification de la copie modifiera aussi l'original.
   • On rappelle qu'il existe un module appelé `copy` avec une fonction `deepcopy` qui résout ce problème des copies.

b) Pour nous, par commodité, on pourra se servir plutôt des `np.array` de `numpy` qui ont des fonctions d'initialisations intégrées comme `np.zeros((n,n))` qui initialise un `np.array` rempli de 0.

1.3 Les tableaux NUMPY : `np.array`

Dans toute la suite, on considère qu'on a importé `numpy` via :

```
import numpy as np
```

La syntaxe pour déclarer un tableau NUMPY codant la matrice donnée au début du cours est très proche :

```
A=np.array([[1,2],[3,4]])
```

Cette fois, on a un affichage en deux dimensions :

```
array([[1, 2],
       [3, 4]])
```

On accède aux entrées de deux manières : `A[i][j]` ou `A[i,j]`

Pour initialiser une matrice 3×2 remplies de zéros :

```
A=np.zeros((3,2)) # noter les doubles parenthèses.
print(A)
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

Noter que par défaut, ces zéros sont des flottants mais :

```
A=np.zeros((3,2),dtype=int) # dtype pour data type
```

fabriquera une matrice dont les entrées sont des entiers.

2 Résolution des systèmes triangulaires :

On considère un système $Y = TX$ avec $T \in TS_n(\mathbb{K})$ inversible et $Y \in M_{n,1}(\mathbb{K})$ fixés, et on cherche l'unique solution $X \in M_{n,1}(\mathbb{K})$ de ce système.

Un tel système se résout *de bas en haut.*

A la ligne L_n, on va seulement faire un quotient pour trouver $x_n = y_n/t_{n,n}$.

Mais ensuite à la ligne L_i, on obtiendra x_i par la formule :

$$x_i = \frac{1}{t_{i,i}}(y_i - \sum_{j>i} t_{i,j}x_j).$$

Questions à rédiger à la maison (sur fichier .py)

a) Donner, en justifiant votre réponse, l'ordre de grandeur de la complexité de cet algorithme de résolution :

$$\Theta(n), \Theta(n^2), \Theta(n^3) \,?$$

b) Ecrire une fonction PYTHON appelée `resoutTS` qui prend comme arguments une matrice triangulaire supérieure inversible $T \in TS_n(\mathbb{K})$ et une matrice $Y \in M_{n,1}(\mathbb{K})$ et qui renvoie l'unique solution $X \in M_{n,1}(\mathbb{K})$ du système $Y = TX$. (Ici les nombres dans $\mathbb{K}$ seront des flottants).

3 Les opérations simples sur les lignes d'une matrice

Les programmes suivants seront écrits pour s'appliquer même à une matrice rectangulaire.

a) Ecrire une fonction `echangeLI(A,i,j)` qui échange les lignes `i` et `j` de la matrice `A`,

b) Ecrire une fonction `transvectionLI(A,i,j,mu)` qui dans la matrice `A` fait l'opération $L_i \leftarrow L_i + \mu L_j$.

On tient compte du fait que les `array` sont des objets mutables :

Les fonctions précédentes modifieront la matrice `A` passée en argument.

Remarque : *Pourquoi ai-je appelé la constante* `mu` *et pas* `lambda` *? Parce qu'en* PYTHON *le nom* `lambda` *est réservé pour la déclaration de fonction en une ligne.*

4 Présentation du pivot : mise sous forme triangulaire

On fixe une matrice $A \in M_n(\mathbb{K})$ *inversible.*

4.1 La méthode du pivot, avec une spécificité pour les machines

Le premier pivot

Au début de la méthode du pivot, on veut « nettoyer » la première colonne en utilisant $A(1,1)$ comme pivot, via $L_i \leftarrow L_i - \mu_i L_1$ pour $i \geq 2$ où $\mu_i = A(i,1)/A(1,1)$. Mais le (seul) problème est que $A(1,1)$ peut être nul.

Ce que dirait un mathématicien :

On doit donc d'abord rechercher la première entrée non nulle dans la colonne 1 et une fois cette entrée trouvée à une ligne L_i échanger L_i et L_1. Comme A est inversible, on sait qu'on va toujours en trouver, car la première colonne ne peut pas être nulle.

Une précaution liée au calcul numérique pour le choix du pivot :

• Quand on manipule des flottants, le test de *l'égalité* à zéro n'est pas adapté, car parfois dans les calculs un nombre qui vaudrait théoriquement zéro sera remplacé par un nombre très petit. On pourrait remplacer le test d'égalité à zéro par la comparaison avec le epsilon machine, mais :

• pour des raisons de *meilleure précision* du calcul numérique,

il vaut mieux éviter de diviser par des très petits nombres.

Corollaire du principe précédent, ce que dit un mathématicien numérique :

il vaut mieux choisir comme pivot dans la colonne 1, *l'entrée* ayant la plus grande valeur absolue

Le pivot à l'étape i :

A l'étape i, en suivant le principe précédent, on choisit *parmi les lignes L_j avec $j \geq i$*, celle où l'entrée $A(j,i)$ a la plus grande valeur absolue, et on l'échange avec L_i. Ensuite on se sert de cette nouvelle L_i pour nettoyer la colonne C_i en dessous de la diagonale.

Remarque du mathématicien : là encore, on est sûr qu'une des entrées $A(j,i)$ avec $j \geq i$ est non nulle. Sinon, comme à ce stade les premières colonnes $C_1, \ldots, C_{i-1}$ sont déjà celles d'une matrice T.S., on aurait $C_1, \ldots, C_i$ liées.

4.2 Estimation de la complexité de la méthode du pivot précédente

Ecrite de manière plus formelle, la description de l'algorithme du pivot donnée au paragraphe précédent devient :

On numérote ici les lignes en $L_0, \ldots, L_{n-1}$ dans l'esprit de PYTHON.

```
pour i de 0 à n-2 :
        trouver j >=i  tel que |A(j,i)| soit maximum
        échanger L_i et L_j
        pour k de i+1 à n
             L_k <- L_k - mu_k L_i
```

Pour chaque valeur de $i \in [\![0, n-2]\!]$:

- la recherche de j coûte $n-i$ comparaisons,
- l'échange éventuel de L_i et L_j coûte $2n+2$ affectations ($2n$ pour les entrées des lignes, et 2 pour i et j)
- pour chaque valeur de k entre $i+1$ et $n-1$, la transvection coûte n affectations, et autant de divisions, multiplications, soustractions (une par entrée de la ligne).

Propriété En ajoutant toutes ces contributions comme comptant 1, on obtient pour la méthode du pivot sur une matrice $A \in GL_n(\mathbb{K})$ un coût en $\Theta(n^3)$.

4.3 Mise en oeuvre de la méthode du pivot : à vous de jouer

Travail à faire : • écrire une fonction `devient_triangle` qui prend comme argument une matrice `A` *qui ne sera pas modifiée par la fonction* et qui renvoie une matrice `T` triangulaire supérieure obtenue par pivot sur les lignes à partir de `A`.

Outils utiles : • On pourra créer d'abord une petite fonction `cherche_pivot` auxiliaire, qui recherche la plus grande entrée non nulle en dessous de la diagonale dans une colonne `j`.

5 Application à la résolution de systèmes de Cramer

5.1 Codage matriciel d'un système *quelconque* avec second membre :

Remarque : Si on se donne un système linéaire quelconque $S : Y = AX$ avec $A \in M_{m,n}(\mathbb{K})$, $Y \in M_{m,1}(\mathbb{K})$ et $X \in M_{n,1}(\mathbb{K})$, où A et Y sont fixés et on cherche X, si on transforme ce système S par des opérations élémentaires sur les lignes, on arrive à un système équivalent $S' : Y' = A'X$ où Y' et A' ont été modifiées par les *mêmes* opérations opérations sur les lignes.

Ces systèmes équivalents ont bien sûr même ensemble de solution. On en déduit la :

Propriété-définition de la matrice augmentée associée à un système :

Toutes les opérations sur un système peuvent être codées sur ce qu'on appelle la *matrice augmentée* de ce système à savoir la matrice $(A|Y)$ obtenue en rajoutant Y comme dernière colonne à A. Les opérations faites sur lignes du système sont faites sur les lignes de cette matrice. On arrive à une matrice $(A'|Y')$ telle qu'on ait toujours l'équivalence : $AX = Y \Leftrightarrow A'X = Y'$.

Exemple : On considère le système

$$\begin{cases} x+y+7z &= -1 \\ 2x-y+5z &= -5 \\ -x-3y-9z &= -5 \end{cases}$$

A ce système, on associe la matrice augmentée :

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 2 & -1 & 5 & -5 \\ -1 & -3 & -9 & -5 \end{pmatrix}$$

Comparons alors l'effet d'une opération élémentaire sur le système et la matrice augmentée. Par exemple, avec : $L_2 \leftarrow L_2 - 2L_1$

- le système devient :

$$\begin{cases} x+y+7z &= -1 \\ -3y-9z &= -3 \\ -x-3y-9z &= -5 \end{cases}$$

- la matrice augmentée devient :

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & -3 & -9 & -3 \\ -1 & -3 & -9 & -5 \end{pmatrix}$$

Ceci illustre bien qu'il suffit de conduire les opérations élémentaires sur la matrice augmentée pour coder les différents systèmes équivalents obtenus par opérations élémentaires.

Ici, en poursuivant le pivot avec la matrice augmentée : $L_3 \leftarrow L_3 + L_1$

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & -3 & -9 & -3 \\ 0 & -2 & -2 & -6 \end{pmatrix}$$

$L_2 \leftarrow \dfrac{-1}{3} L_2$

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & 1 & 3 & 1 \\ 0 & -2 & -2 & -6 \end{pmatrix}$$

$L_3 \leftarrow L_3 + 2L_2$

$$\begin{pmatrix} 1 & 1 & 7 & -1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 4 & -4 \end{pmatrix}$$

Cette matrice augmentée *dont la première partie est une matrice TS* 3×3 code le système triangulaire équivalent à notre système initial :

$$\begin{cases} x+y+7z &= -1 \\ y+3z &= 1 \\ 4z &= -4 \end{cases}$$

5.2 Les deux étapes résolution d'un système de Cramer par le pivot

Définition : Un système de Cramer est un système $AX = Y$ avec A matrice carrée *inversible* donnée, Y colonne donnée, et d'inconnue une colonne X. On sait donc que ce système admet un unique solution.

La première étape de la résolution :

Transformer A en une matrice A' triangulaire sup. avec la méthode du pivot sur les lignes donnée au § 4.1. Appliquer les mêmes transformations sur les lignes à Y ce qui donne une nouvelle colonne Y'. (On peut utiliser la matrice augmentée pour cela).

La deuxième étape de la résolution :

Résoudre le système triangulaire $A'X = Y'$ avec la méthode du § 2.

Question : Quelle est la complexité totale de cet algorithme ?

5.3 Mise en oeuvre de cette méthode : à vous de jouer

a) Première étape : à l'aide de ce qui précède, en modifiant légèrement la fonction `devient_triangle` précédente, écrire une fonction qui ramène un système `Y=AX` à un système triangulaire.

Pour cela, il suffit de faire subir à `Y` les mêmes opérations que celles faites sur `A`.

Ceci se formalise aussi avec la notion de *matrice augmentée :* $[A|Y]$

b) Deuxième étape : écrire une fonction `resout_systeme` qui prend comme argument une matrice inversible `A` et une colonne `Y` et renvoie l'unique `X` tel que `Y=AX`.

6 Cas des matrices à coefficients entiers ou rationnels

Question : Comment utiliser les algorithmes précédents pour faire du calcul exact sur les rationnels ?

Réponse : Il suffit de déclarer des matrices `A` et `Y` à coefficients rationnels.

On utilisera le module `fractions` comme suit :

```
>>> from fractions import * # ce qui suit est un exemple de calcul :
>>> Fraction(1,3)+Fraction(1,2)
Fraction(5, 6)
```

a) Vérifier que vos fonctions précédentes s'appliquent bien à des matrices dont les entrées sont des `Fractions` en renvoyant des `Fractions`.

b) **(Par commodité pour la suite) :** écrire une fonction `convert_ratio` qui prend une matrice `A` rectangulaire quelconque dont les coefficients sont supposés entiers et renvoie *une autre matrice* où on a converti chaque entrée `A[i][j]` en la fraction `Fraction(A[i][j],1)`. Ecrire aussi `convert_float` qui fait l'inverse et convertit les entrées en flottant.

7 Calcul exact sur les rationnels vs. calcul sur les flottants : la matrice de Hilbert

On considère la matrice dite de Hilbert $H_n = \left(\frac{1}{i+j+1}\right)_{(i,j)\in[\![0,n-1]\!]}$.

Cette matrice (qui intervient notamment dans certains problèmes d'analyse) est très connue notamment pour :

- être inversible : son inverse est même à coefficients *entiers* et on peut donner une formule explicite pour cet inverse avec des binomiaux...
- avoir un calcul d'inverse très *instable* du point de vue du calcul numérique.

L'exercice qui suit va illustrer ces affirmations :

a) Ecrire une fonction `hilbert(n)` qui fabrique la matrice de Hilbert H_n en PYTHON comme une matrice à coefficients de type `Fraction`.

b) Appliquer la fonction `devient_triangle` à H_n pour $n = 20$: l'entrée la plus petite de H_n est $1/(2n-1)$. Commentez la taille des dénominateurs obtenus pour `T=devient_triangle(hilbert(20))`

c) A l'aide de `convert_float(T)`, que peut-on dire des entrées diagonales de la matrice `T` ? Que penser *a priori* des risques de l'algorithme de résolution appliqué à un système $TX = Y$?

d) On veut expliciter la première colonne de la matrice H_n^{-1}. Pour cela il suffit de résoudre le système $H_n X = E_1$ où E_1 est la première colonne canonique.
Résoudre ce système à l'aide de la fonction `resout_systeme` avec H_n codée avec des `Fractions`.

e) Résoudre ce système à l'aide de la fonction `resout_systeme` avec H_n codée avec des `floats`.

f) Que dire des résultats obtenus ?